Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 1783 – 1788

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Hit Me: Blackjack as a Sure Bet for Teaching Algorithmic Skills

Timothy M. Walker*

Division of Mathematics, Computer, and Natural Sciences, Ohio Dominican University, Columbus, OH, 43219, USA

Abstract

Introductory computer science courses need to cover fundamental concepts and terminology as well as algorithmic skills. One way to accommodate these demands is to employ problem-based assignments as a vehicle for the programming content. An assignment is discussed in which students must develop a blackjack application in both *Excel* and *Mathematica*. Common solutions are considered in relation to general problem-solving strategies, and the assignments are contrasted with full-semester problem-based learning approaches.

Keywords: Problem-based learning; Excel; Mathematica; blackjack

1. Introduction

As in many fields, undergraduate students entering the fields of Computer Science, Computer Information Systems, Information Technology, etc., typically start their curriculum with an introductory course which introduces them to the fundamental language and concepts of their major. However, computational fields are perhaps a bit unique in that an introduction to the discipline should include an exposure to some of the basic skills (i.e., programming) as well as vocabulary and concepts. This is not to suggest, of course, that there are not many other fields which must develop important specialized skill-sets, but oftentimes that training can be concentrated in subsequent upper-level courses, after the student has some basic familiarity with the scope of the field itself. In CS fields, though, programming concepts are fundamental enough that they cannot be separated from a descriptive survey of the discipline.

This, then, presents a challenge for introductory CS courses. It can be difficult to present algorithmic techniques without additional (and time-intensive) explanation of programming syntax. Conversely, traditional textbook coverage of programming concepts (e.g., variable naming conventions, data structures, debugging techniques, indexing and looping, etc.) are potentially valuable for the beginning programmer but may not be fully appreciated when they are divorced from an application of the skill. This suggests that the timing of course content might be as important as the content itself; there is a synergy inherent in the coordination of programming projects with accompanying programming concepts. Such an approach is consistent with the problem-based learning (PBL)

* Corresponding author: Timothy M. Walker. Tel.: +6142514658.
E-mail address: walkert@ohiodominican.edu

doi:10.1016/j.procs.2012.04.196

model, in which course content is manifested through student-directed inquiry [1,2,3,4,5,6,7].

It is also the case, however, that programming is only one topic within a general introductory course. A survey course is not intended to take the place of a first course in programming, and it would not be efficient to duplicate the amount of effort that it takes to fully immerse a student into a traditional programming environment (i.e., a Java or C++ compiler and IDE). Therefore, projects within a more general course should be manageable within a few weeks and should probably be developed in a high-enough level environment to keep the focus on algorithmic issues rather than syntactic structure. At the same time, the projects should be challenging, and hopefully relevant enough to the students to foster the motivation to work independently outside of class.

In particular, the environment should be accessible to students if they are to have any hope of working on their own. There are two obvious possibilities for this: it could be an environment that is in some way already familiar to the students; or it could be an environment that is sufficiently user-friendly, and with a sufficiently transparent help system, to be usable with a minimum of introduction. Each of these possibilities has its own benefits and drawbacks.

The projects described here are used within a freshman-level course entitled Computer Information Systems (CIS) 180 – Survey of Computer Information Systems. The course is intended to capture CIS majors in their first-semester freshman year, to introduce them to their chosen field of study and also to provide an overview of topics that will be covered in greater depth later in their undergraduate career. The projects are assigned to pairs of students, and consist of the implementation of a simple interactive blackjack game. The assignment is given twice; once within *Excel* and once within Wolfram *Mathematica*.

2. The blackjack assignment

Blackjack was chosen for this assignment for a variety of reasons: it is familiar to the majority of students; for many students, there is an intrinsic interest in games, so there is more motivation than for an arbitrary assignment; the basic rules of blackjack are fairly simple compared to many other games; and there are a number of additional rules that can be implemented which extend the complexity of the game. The assignment is to implement a system which will play the part of the dealer in a blackjack game. It should be able to deal the cards from a deck to both itself and to the player, each getting one card up (showing) and one card down (hidden). The player, seeing both of his or her cards and the dealer's up card, then decides to stand (keep the hand as is) or hit (take another card). If the player hits too many times and exceeds a total of 21, he or she "busts" and loses the hand. If the player stands (without busting), the dealer then must hit if it has less than 17 and must stand otherwise.

These rules provide a challenging but solvable task for the students. They must find a way to represent the basic aspects of the problem-space, including the deck and the separate hands, and must also navigate the logic of determining a winner, since there are multiple ways to win a hand and multiple ways to lose. Assuming they successfully implement the simple version of the game, there are some additional potential features:

- In general, face cards are worth 10 points each, aces are worth 11 points, and all other cards are worth their numerical value. However, aces can also be worth 1 point if the player would otherwise bust.
- Since blackjack is a gambling game, it is useful to be able to monitor a series of games within a single playing session, so that bets can be placed on each and winnings and losses can be tracked against a virtual account.
- In actual casinos, blackjack players can modify their hands and bets by "splitting" or "doubling-down" on certain hands.
- Typically there are multiple players, each independently playing against the dealer.
- The dealers usually use multiple decks shuffled together, rather than just one.

The net effect of these rules is that students can work in a modular fashion, first solving and troubleshooting the core problems and then moving on to additional features. It is also the case that the initial step of defining the problem-space greatly affects the way in which almost all other decisions are made, so they must spend some time trying (and rejecting) various possible strategies before proceeding to the logic of hitting and standing. Thus, it is a good exercise in general problem-solving. The assignments are given to groups of two to encourage sharing of ideas and potential approaches.

2.1. Excel

The first blackjack assignment is given within *Excel*, mostly because the students all have a fair degree of familiarity with it; they are able to begin working with a minimum of introduction. However, there is a significant disadvantage to *Excel* – since it is not a real programming environment, with real variables or control structures, students must invent ways to represent features of the problem. A common solution is to list the 52 card values in the deck within a column. To randomly deal from the deck, a random cell address can be generated in order to reference a card value. However, this method makes it difficult to keep track of the cards that have already been dealt (i.e., random sampling without replacement). A different method, which more directly models the act of shuffling, is to use a second column, consisting of randomly-generated numbers. The two columns can then be sorted on the basis of the random column, and dealing simply consists of progressively referencing the card column in order.

In the absence of an obvious output format in *Excel*, there are two different types of interfaces to which students gravitate. One is to use the standard spreadsheet view of columns and rows, using some form of formula filling as the mechanism for dealing additional cards. An example of this is shown in Figure 1. An alternative is to build a graphical interface within the spreadsheet, such as in Figure 2. Note that, in this case, the graphics are created from ordinary spreadsheet elements (using background colors, symbols, cell dimensions, justification, etc.) rather than embedded graphics. The unspecified nature of the assignment lends itself to discovery of formats and features within the application.

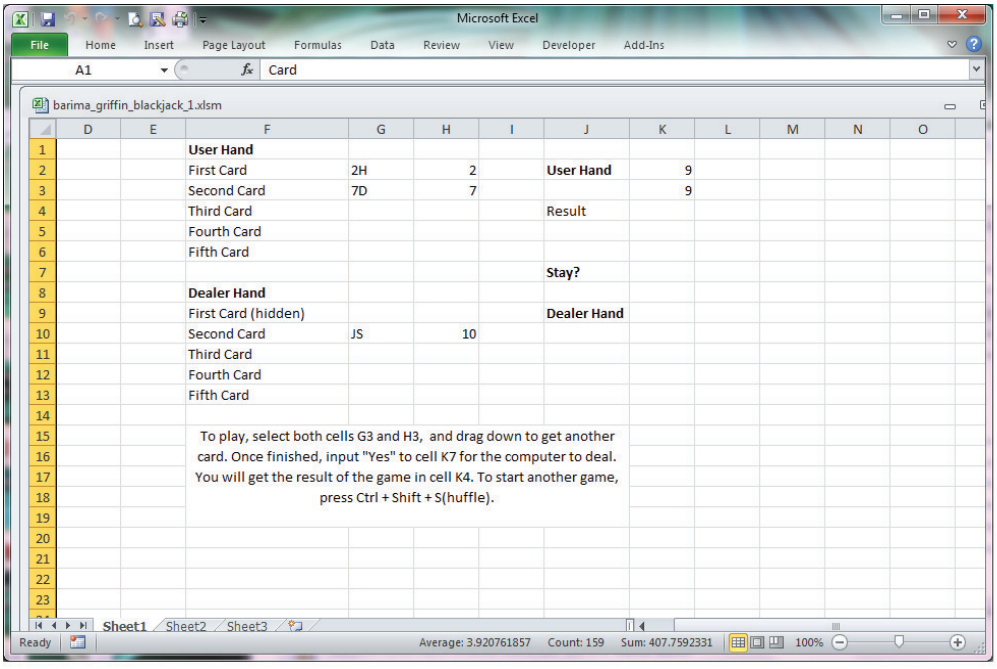

Fig. 1. Column/row textual interface

In general, the guiding principle is to assign the task first and then provide feedback and information as students determine for themselves that they need it, rather than to attempt to provide the relevant information first as a disconnected lecture. Students try out different ideas as they work, and learn important techniques such as absolute and relative referencing, conditional formatting, use of specialized functions, and macros. More significantly, though, they learn about problem-solving strategies and the value of choosing appropriate tools for a given task.

2.2. Mathematica

Students find that working with *Mathematica* gives an opposite pattern of advantages and disadvantages. Unlike *Excel*, they have typically had no exposure to *Mathematica*, so everything about the syntax and user interface is brand new. However, they realize, after having tried to implement complex logical conditions in *Excel*, the value of having a true programming environment at their disposal. Therefore they are able to avoid getting too distracted by the syntax, and can focus on their goal of solving the problem. *Mathematica* also has a few advantages as an introductory platform; it uses a functional language in which the parameters are listed very similarly to *Excel*, it doesn't require initial declarations or type identification of variables, and the help system is fairly robust. Additionally, *Mathematica* has a natural language processing feature that is unique among scientific computing platforms; students can type commands in plain English, such as "list all numbers from 1 to 52," and it will attempt to replace the text with the appropriate structured command. Even if the translation doesn't produce the intended result, this can be a helpful way for students to explore the environment and learn about the syntax.

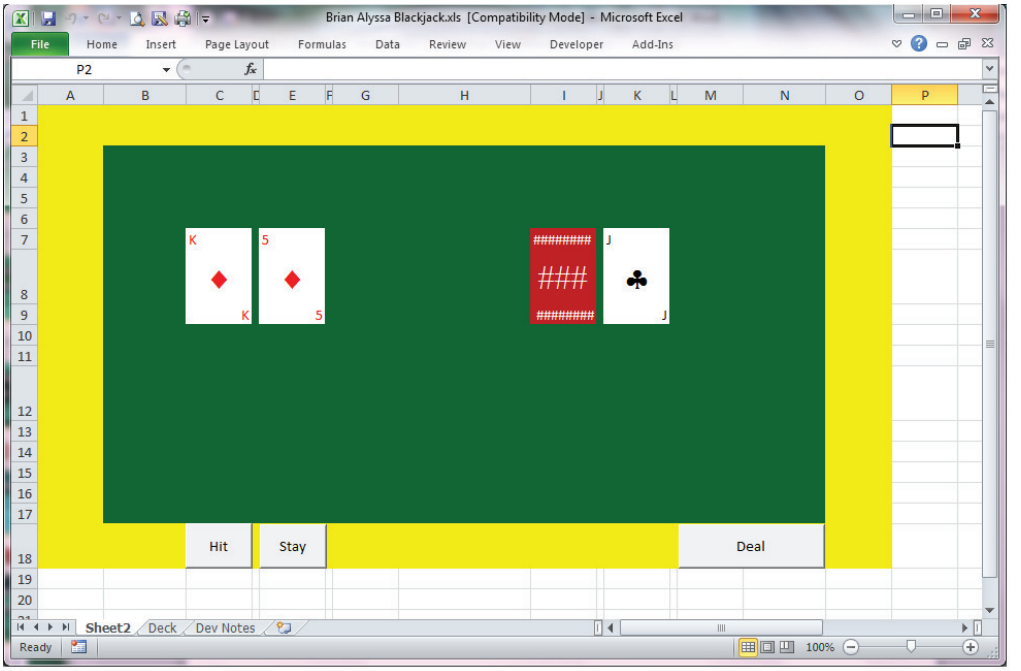

Fig. 2. Graphical interface

Mathematica also has a rich set of interactive control elements which students often incorporate into their solutions. A typical strategy is to structure the deck as an array (or, in *Mathematica* terminology, "list") and use buttons or similar controls to randomize and systematically sample from the list. The output is sometimes textual, but students also may use graphical representations of cards, as shown in Fig. 3. One of the benefits of the assignment is that many students develop multi-dimensional arrays, with card names, values, suits, and potentially images stored in parallel. This forces them to learn how to index and address the dimensions appropriately. Within the context of their blackjack program, these concepts are learned organically and without difficulty, whereas an abstract discussion of multi-dimensional arrays is typically much less accessible.

3. Summary and conclusions

Overall, the blackjack assignments offer a compelling example of the value of context when learning a new skill. The course described here is not a classic example of problem-based learning in its purest sense, because many PBL

adherents define it at the course or program level, rather than individual assignments [e.g., 2,3,4,5,7]. However, these assignments certainly contain many of the fundamental characteristics of PBL. For the Survey of CIS course, it is advantageous to use a blended approach, with traditional methods used for the terminology content and PBL techniques for the problem-solving and algorithmic development skills.

Like many domains, programming skills are more easily learned when they are embedded within an engaging context [2,6,7,8]. Motivation is typically much higher on the blackjack assignments than on other tasks, with students often lingering after class to ask follow-up questions or to ask for specific examples. They also work more independently in terms of finding information from help systems and online resources, and are usually very willing to share new tips and techniques with fellow classmates. As a result, they learn more efficiently and seem to develop a better comprehension of concepts (e.g., data structures, logical comparisons, looping) than they otherwise would in a more traditional exposition of the same topics. In this class, the goal is less about specific programming skill than general problem-solving skills, so the ability to compare and contrast implementations in *Excel* and *Mathematica* is valuable. There is no better way to appreciate the scope of a particular tool than to try to use it for a task it was not designed for, as students soon learn when struggling with multiple logical conditions in *Excel*. For all of these reasons, the problem-based component has been an extremely beneficial aspect to the survey course.

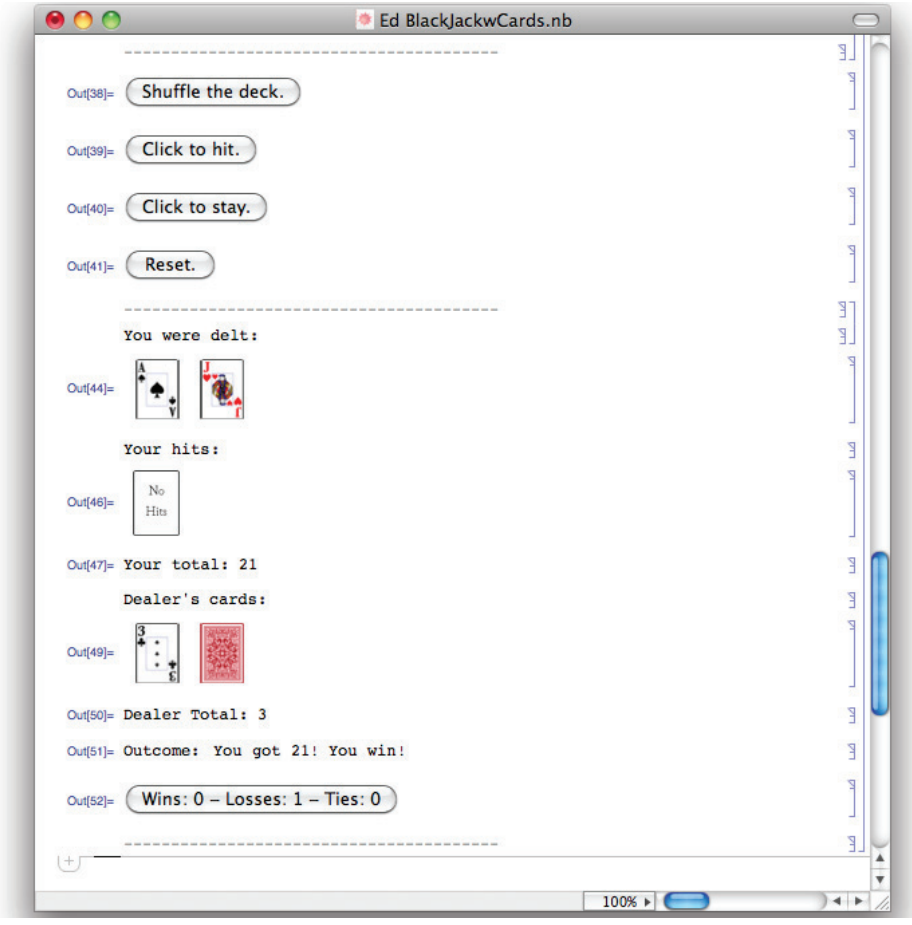

Fig. 3. *Mathematica* output

References

1. M. A. Albanese and S.A. Mitchell, Problem based learning: A review of literature on its outcomes and implementation issues. Academic Medicine 68 (1993) 52-81.

2. M. Barg, A. Fekete, T. Greening, O. Hollands, J. Kay, and J.H. Kingston, Problem-based learning for foundation computer science courses. Computer Science Education 10(2) (2000) 109-128.

3. M.A. Dahlgren and L.O. Dahlgren, Portraits of PBL: students' experiences of the characteristics of problem-based learning in physiotherapy, computer engineering and psychology. Instructional Science 30 (2002) 111-127.

4. C.E. Hmelo-Silver, Problem-based learning: What and how do students learn? Educational Psychology Review 16(3) (2004) 235-266.

5. W. Hung, D.H. Jonassen, and R.Liu, Problem-based learning. In J.M. Spector, M.D. Merrill, J. van Merrienboer, and M.P. Driscoll (eds.) Handbook of research on educational communications and technology, New York: Routledge (2008) 578-593.

6. J. O'Kelly and J.P. Gibson, RoboCode & problem-based learning: A non-prescriptive approach to teaching programming. ACM SIGCSE Bulletin 38(3) (2006) 217-221.

7. A. Soares, Problem based learning in introduction to programming courses. J. Computing Sciences in Colleges 27(1) (2011) 36.

8. C.C. Bareiss, A semester project for CS1. ACM SIGCSE Bulletin 28(1) (1996) 310-314.